

A New Kernelized Associative Memory and Some of Its Applications

Matthew Saltz and Lluís A. Belanche¹

Abstract. The classical Bidirectional Associative Memory (BAM) allows for the storage of pairs of vectors, such that when either member of the pair is presented to the BAM, the other member may be successfully recalled. This work presents a novel BAM, improved with respect to its capacity and noise performance through the use of the kernel trick, a common technique in machine learning for transforming linear methods into nonlinear methods. By kernelizing the BAM's energy function directly and defining new methods for recall, the kernel BAM shows improved performance compared to both the original BAM as well as a previously existing nonlinear BAM. This is demonstrated with thorough experimentation on synthetic datasets, and several practical applications are given on real data.

1 INTRODUCTION

Associative memories (AMs), designed to store items (known as *patterns*) and then recall them even under noisy conditions, have been studied for decades [1, 2, 3]. The kind of recall performed by an AM is trivial for the human brain, and many AMs are biologically inspired and typically represented as artificial neural networks. Thus, AMs are often interesting from both a biological modeling as well as a machine learning perspective –see, e.g., [4, 5, 6, 7]. AMs are judged based on the number of patterns they can store without sacrificing their recall abilities (known as the *capacity*), as well as the average amount of noise that can be added to a stored pattern while still maintaining effective recall (known as the *noise performance*).

A classical example of an AM neural network is the Bidirectional Associative Memory, or BAM, which stores *pairs* of associated items [2]. When one item from a pair stored in the BAM is presented to the network, the network retrieves and returns its associated item, even if the input has been corrupted in some way (*i.e.*, it contains noise). An interesting property of the BAM is that recall may occur in both directions, either recalling the first or second item of a pair.

One limitation of the BAM (and other AM networks as well) is that when the patterns to be stored are linearly dependent, recall performance suffers; and as more patterns are added to the network, they will inevitably become linearly dependent, and so the capacity of the BAM is ultimately limited [2, 8, 7]. In order to alleviate this problem, it is possible to project the input patterns into a higher dimensional space, where their representations are more likely to be orthogonal. The larger the dimensionality, the higher the capacity of the network becomes. Recall could be performed in this space, and the result could be mapped back to the input space.

In this work, a novel, improved BAM is presented that takes advantage of the kernel trick to improve capacity and noise performance. While others have used the kernel trick for a variety of au-

toassociative memories (which store only a set of patterns rather than a set of pairs of patterns), the focus has been primarily on kernelizing the recall method directly [9, 10, 4, 11] or, if not, on using the kernel AM for another purpose, such as classification [8]. The kernel BAM presented here focuses on maintaining as closely as possible the analogy with the original BAM by kernelizing the associated *energy function* of the model, and then deriving novel methods for recall from there. Thorough experimentation demonstrates the superiority of the kernel BAM in a variety of cases over the original BAM as well as the most analogous nonlinear BAM found in the literature.

2 ENERGY-BASED MODELS

Given a set of p pairs $\mathcal{S} = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \dots, (\mathbf{x}^{(p)}, \mathbf{y}^{(p)})\}$ with $\mathbf{x}^{(i)} \in \mathcal{X}$, $\mathbf{y}^{(i)} \in \mathcal{Y}$, the goal of an AM is to store the pairs in such a way that, when presented with a vector, known as a *pattern*, $\mathbf{x}^{(i)}$, its corresponding pattern $\mathbf{y}^{(i)}$ is correctly recalled, even if the input pattern $\mathbf{x}^{(i)}$ has been corrupted in some way. An example of this would be to store (*name, telephone*) pairs so that, when a user wants to recall the telephone of someone \mathbf{x} , the name of this someone \mathbf{x} could be presented to the memory, which would recall the correct telephone number \mathbf{y} even if the name \mathbf{x} –as presented to the AM– is corrupted by background noise. The two primary metrics for the quality of an AM are *capacity*, which describes the number of patterns (or pairs) that can be stored in the AM with respect to the dimensionality of the patterns, and *noise performance*, which is how well the AM can recover memories in the presence of noise.

The scenario above, where the AM stores pairs of patterns, is known as an *heteroassociative* memory. The special case where the AM stores only a set of single patterns to remember, rather than a set of pairs, is known as the *autoassociative* case. Some AMs are specially designed for this case, like the well-known Hopfield network [1]. The autoassociative case is equivalent to having $\mathbf{x}^{(i)} = \mathbf{y}^{(i)}$ for all $i \in \{1, \dots, p\}$ in the formulation above.

In order to memorize the set \mathcal{S} of associated pairs, we assume there is some general dependency between the components x_i of \mathbf{x} and the components y_j of \mathbf{y} . We then want to discover the form of this dependency so that we can use it to map an input vector $\mathbf{x}^{(i)}$ from \mathcal{S} to its corresponding vector $\mathbf{y}^{(i)}$ (and potentially vice versa). One way to do this is to define first the believed structure, or *architecture*, of the dependencies, and then introduce an associated function, known as the *energy function*, that takes \mathbf{x} and \mathbf{y} as inputs and evaluates how well \mathbf{x} and \mathbf{y} satisfy the dependency. Finally, we need a way to *infer* one from the other. This process of defining an architecture, an associated energy function, a training method/loss function, and an inference method is known as *energy-based modeling*, and several kinds of AMs can be described naturally in this framework, including

¹ Technical University of Catalonia, Spain, email: belanche@cs.upc.edu

the BAM –see [12] for a review on energy-based learning models.

The most basic form of an AM is the *linear* AM (LAM). When \mathbf{x} and \mathbf{y} are bipolar vectors, with $\mathbf{x} \in \{-1, +1\}^m$ and $\mathbf{y} \in \{-1, +1\}^n$, one defines the associated energy function

$$E(\mathbf{x}, \mathbf{y}) = -\frac{1}{2} \mathbf{y}^T \mathbf{W} \mathbf{x}, \quad \mathbf{W} \in \mathbb{R}^{n \times m} \quad (1)$$

where lower energy values are associated with more compatible vectors; assuming \mathbf{x}_0 as input, we may infer the most compatible \mathbf{y} by taking

$$\mathbf{y}^* = \arg \min_{\mathbf{y} \in \{-1, +1\}^n} E(\mathbf{x}_0, \mathbf{y}) = \text{sgn}[\mathbf{W} \mathbf{x}_0] \quad (2)$$

where $\text{sgn}[\cdot]$ returns the vector of signs of its argument. In classical *Hebbian learning*, \mathbf{W} is computed as the sum of correlation matrices between the $\mathbf{x}^{(i)}$ and their corresponding $\mathbf{y}^{(i)}$, that is,

$$\mathbf{W} = \sum_{i=1}^p \mathbf{y}^{(i)} (\mathbf{x}^{(i)})^T = \mathbf{Y} \mathbf{X}^T \quad (3)$$

being $\mathbf{X} \in \mathbb{R}^{m \times p}$ the matrix where the i -th column is $\mathbf{x}^{(i)}$, and $\mathbf{Y} \in \mathbb{R}^{n \times p}$ the matrix where the i -th column is $\mathbf{y}^{(i)}$. Another approach solves the equation $\mathbf{Y} = \mathbf{W} \mathbf{X}$ to get $\mathbf{W} = \mathbf{Y} \mathbf{X}^\dagger$, where $\mathbf{X}^\dagger = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T$ is the Moore-Penrose pseudoinverse. This method has a better capacity and noise performance in general, since Hebbian learning suffers when vectors are correlated [7, 11].

3 KERNEL BIDIRECTIONAL ASSOCIATIVE MEMORIES

The LAM above is limited in modeling capability and in its ability to accurately recover memories with noisy inputs [7]. One way to improve this is to introduce a recurrent structure, feeding the output of the network in one step back into the network as input in the next step. In this way, the network can iteratively improve on its guess for the output. One such recurrent AM is the Bidirectional Associative memory (BAM). The BAM may use the same energy function as in Eq. (1), but in contrast with the LAM, which can only recall in one direction (retrieving the \mathbf{y} for a given \mathbf{x}), the BAM can recall in both directions. It accepts \mathbf{x} or \mathbf{y} as input and performs consecutive updates until the state of the network no longer changes.

The goal of the update process is to start at the input and continually decrease the value of the energy function until it finds a local minimum. The recurrent update process is as follows, where $\mathbf{s}_x(t)$ and $\mathbf{s}_y(t)$ indicate the state of the \mathbf{x} and \mathbf{y} portions of the network at update step t , respectively. Assuming that the network is initially presented with an input for \mathbf{x} :

$$\mathbf{s}_x(0) = \text{input } \mathbf{x}$$

$$\mathbf{s}_y(1) = \arg \min_{\mathbf{y} \in \{-1, +1\}^n} E(\mathbf{s}_x(0), \mathbf{y}) = \text{sgn}[\mathbf{W} \mathbf{s}_x(0)]$$

$$\mathbf{s}_x(2) = \arg \min_{\mathbf{x} \in \{-1, +1\}^m} E(\mathbf{x}, \mathbf{s}_y(1)) = \text{sgn}[\mathbf{s}_y(1)^T \mathbf{W}]$$

...

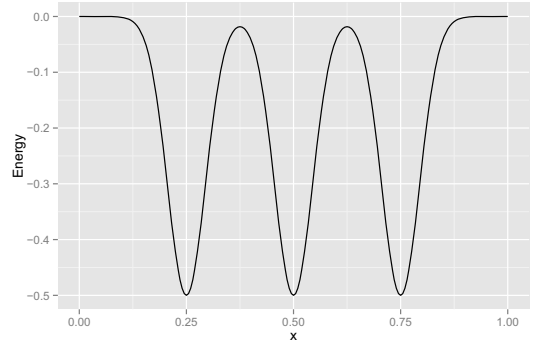
$$\mathbf{s}_y(t+1) = \arg \min_{\mathbf{y} \in \{-1, +1\}^n} E(\mathbf{s}_x(t), \mathbf{y}) = \text{sgn}[\mathbf{W} \mathbf{s}_x(t)]$$

$$\mathbf{s}_x(t+2) = \arg \min_{\mathbf{x} \in \{-1, +1\}^m} E(\mathbf{x}, \mathbf{s}_y(t+1)) = \text{sgn}[\mathbf{s}_y(t+1)^T \mathbf{W}]$$

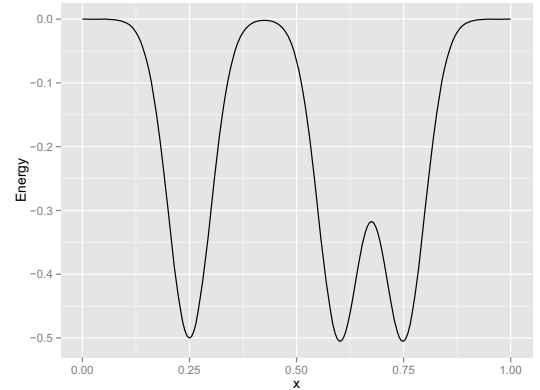
Because there is only a finite number of possible bipolar vectors, and because the energy decreases after every update, this process

must eventually *converge* to a local minimum of the energy function, where convergence means that the state of the network no longer changes state after successive iterations [2].

The BAM network together with the recurrent update process defines a *dynamical system*, and the local minima of the energy function are known as the *attractors* of the dynamical system. Since our goal is to recall memories, we want to train \mathbf{W} such that the stored patterns correspond to the attractors of the system. The space around an attractor in which all vectors will converge to that attractor is known as the *basin of attraction*. Furthermore, we also want to be able to recover memories in conditions of noise. The maximum amount of noise (measured in number of flipped bits) for which an AM performs effectively is known as its *radius of attraction*.



(a) Evenly-spaced memories, at 0.25, 0.5, and 0.75. This illustrates an ideal situation for the radius of attraction, where each memory is an equal distance from the next.



(b) Unevenly-spaced memories, at 0.25, 0.6, and 0.75. The radius of attraction of a BAM with this energy function is limited by the separation between the memories at 0.6 and 0.75.

Figure 1: Two energy functions in the autoassociative case.

An example of two different energy functions with different stored patterns and different radii of attraction is shown in Fig. 1. In this example, we consider the autoassociative case where only \mathbf{x} patterns are stored, with $m = 1$ (so $x \in \mathbb{R}$). To visualize recall for any given x value, one may imagine dropping a ball on the energy function at that x location and watching it roll down to the nearest local minimum. The radius of attraction of a stored pattern describes how close the ball must fall to the pattern in order to successfully converge to that pattern in the worst case. The radius of attraction of an AM is the minimum of all of the radii of attraction of its stored patterns. In Fig. 1a, the memories are evenly spaced and the energy function is such that each memory has an equal radius of attraction

r , which is ideal: the radius of attraction of the AM is also r . In contrast, Fig. 1b illustrates a case where the radius of attraction is limited by two memories which are closer to each other than the rest.

3.1 Kernelizing the energy function

Recall the normal BAM energy function in Eq. (1), being $\mathbf{W} \in \mathbb{R}^{n \times m}$ the BAM weight matrix. Let $k_x : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ and $k_y : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$ be kernel functions, where $k_x(\mathbf{r}, \mathbf{t}) = \langle \phi_x(\mathbf{r}), \phi_x(\mathbf{t}) \rangle_{\mathcal{H}_x}$ for some $\phi_x : \mathcal{X} \rightarrow \mathcal{H}_x$, and $k_y(\mathbf{r}, \mathbf{t}) = \langle \phi_y(\mathbf{r}), \phi_y(\mathbf{t}) \rangle_{\mathcal{H}_y}$ for some $\phi_y : \mathcal{Y} \rightarrow \mathcal{H}_y$. Here \mathcal{H}_x and \mathcal{H}_y are the (potentially) different Hilbert spaces associated with the kernels k_x and k_y , and \mathcal{X} and \mathcal{Y} are the domains of \mathbf{x} and \mathbf{y} , respectively.

Now let $\mathbf{W} = \mathbf{U}\mathbf{V}^T$, where $\mathbf{U} \in \mathbb{R}^{n \times l}$ and $\mathbf{V} \in \mathbb{R}^{m \times l}$ are matrices with column vectors \mathbf{u}_i and \mathbf{v}_i respectively. Then we have:

$$\begin{aligned} E(\mathbf{s}_x, \mathbf{s}_y) &= -\frac{1}{2} \mathbf{s}_y^T \mathbf{U} \mathbf{V}^T \mathbf{s}_x \\ &= -\frac{1}{2} (\langle \mathbf{s}_y, \mathbf{u}_1 \rangle, \dots, \langle \mathbf{s}_y, \mathbf{u}_l \rangle)^T (\langle \mathbf{s}_x, \mathbf{v}_1 \rangle, \dots, \langle \mathbf{s}_x, \mathbf{v}_l \rangle) \\ &= -\frac{1}{2} \sum_{i=1}^l \langle \mathbf{s}_y, \mathbf{u}_i \rangle \langle \mathbf{s}_x, \mathbf{v}_i \rangle \end{aligned} \quad (4)$$

Now replacing \mathbf{s}_x by $\phi_x(\mathbf{s}_x)$, \mathbf{v}_i by $\phi_x(\mathbf{v}_i)$, \mathbf{s}_y by $\phi_y(\mathbf{s}_y)$ and \mathbf{u}_i by $\phi_y(\mathbf{u}_i)$, we have the kernelized energy function:

$$\begin{aligned} E(\mathbf{s}_x, \mathbf{s}_y) &= -\frac{1}{2} \sum_{i=1}^l \langle \phi_y(\mathbf{s}_y), \phi_y(\mathbf{u}_i) \rangle_{\mathcal{H}_y} \langle \phi_x(\mathbf{s}_x), \phi_x(\mathbf{v}_i) \rangle_{\mathcal{H}_x} \\ &= -\frac{1}{2} \sum_{i=1}^l k_y(\mathbf{s}_y, \mathbf{u}_i) k_x(\mathbf{s}_x, \mathbf{v}_i) \end{aligned} \quad (5)$$

Letting $\Phi_y(\mathbf{U})$ be the matrix with columns $\phi_y(\mathbf{u}_i)$ and $\Phi_x(\mathbf{V})$ be the matrix with columns $\phi_x(\mathbf{v}_i)$ we see that

$$E(\mathbf{s}_x, \mathbf{s}_y) = -\frac{1}{2} \phi_y(\mathbf{s}_y)^T \Phi_y(\mathbf{U}) \Phi_x(\mathbf{V})^T \phi_x(\mathbf{s}_x) \quad (6)$$

This corresponds to a fully kernelized BAM, where both vectors in an association are projected into their respective feature spaces, and where the weights exist in feature space as well. Figure 2 shows the BAM as it could be seen in feature space. For certain kernels, like the RBF kernel, this corresponds to orthogonalizing the vectors in feature space, so that the performance of Hebbian learning is improved.

Using now Hebbian learning, setting $\mathbf{U} = \mathbf{Y}$ and $\mathbf{V} = \mathbf{X}$ (in which case $l = p$) we obtain:

$$E(\mathbf{s}_x, \mathbf{s}_y) = -\frac{1}{2} \sum_{i=1}^p k_y(\mathbf{s}_y, \mathbf{y}^{(i)}) k_x(\mathbf{s}_x, \mathbf{x}^{(i)}) \quad (7)$$

In the autoassociative case this corresponds to the energy function of the kernel Hopfield network proposed by Caputo et al. [8], though they do not provide mechanisms for inference and use the energy function as a means for classification only. Though it is beyond the scope of this work, it would be possible to find \mathbf{U} and \mathbf{V} through loss-based training using a loss function like the hinge loss or the negative log-likelihood. This would allow the user to choose the number of vectors in the sum by defining the dimensions of \mathbf{U} and \mathbf{V} . However, using Hebbian learning allows direct comparison of the standard BAM to the kernel BAM.

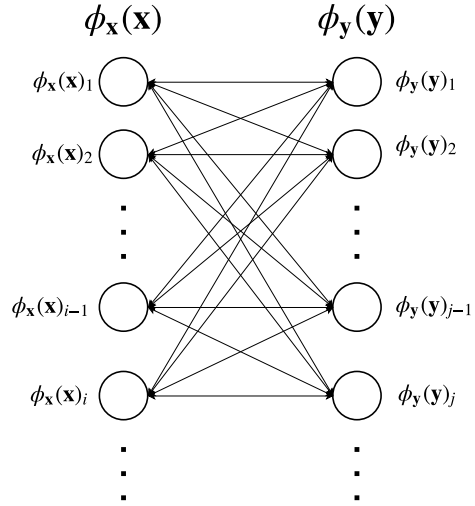


Figure 2: A visualization of the BAM projected into feature space. Note that $\phi_x(\mathbf{x})$ and $\phi_y(\mathbf{y})$ could be infinite-dimensional.

3.2 Inference

Inference (i.e., computing the output for a given input) in the kernel BAM is nontrivial. In a normal BAM update step, we would take

$$\mathbf{s}_y(t+1) = \arg \min_{\mathbf{y} \in \{-1, +1\}^n} E(\mathbf{s}_x(t), \mathbf{y}) = \text{sgn}[\mathbf{W} \mathbf{s}_x(t)] \quad (8)$$

Our first method comes from a comparison with the original BAM update process. Recalling the standard BAM energy function in Eq. (1), we note that $\frac{\partial E}{\partial \mathbf{s}_y} = -\mathbf{W} \mathbf{s}_x$ and we can write the update step as

$$\mathbf{s}_y(t+1) = \text{sgn} \left(-\frac{\partial E}{\partial \mathbf{s}_y}(\mathbf{s}_x(t), \mathbf{0}) \right) \quad (9)$$

Since $\frac{\partial E}{\partial \mathbf{s}_y}$ does not depend on \mathbf{y} , and \mathbf{y} is bipolar,

$$\mathbf{s}_y(t+1) = \arg \min_{\mathbf{y} \in \{-1, +1\}^n} E(\mathbf{s}_x(t), \mathbf{y}) \quad (10)$$

as desired (the math works analogously when updating \mathbf{s}_x). To adapt to the kernel energy function in Eq. (5), we depart from Eq. (9), where

$$\frac{\partial E}{\partial \mathbf{s}_y}(\mathbf{s}_x, \mathbf{s}_y) = -\frac{1}{2} \sum_{i=1}^l \frac{\partial k_y}{\partial \mathbf{s}_y}(\mathbf{s}_y, \mathbf{u}_i) k_x(\mathbf{s}_x, \mathbf{v}_i) \quad (11)$$

which with Hebbian learning finally becomes

$$\mathbf{s}_y(t+1) = \text{sgn} \left(\sum_{i=1}^p \frac{\partial k_y}{\partial \mathbf{s}_y}(\mathbf{0}, \mathbf{y}^{(i)}) k_x(\mathbf{s}_x(t), \mathbf{x}^{(i)}) \right) \quad (12)$$

The intuition behind taking the sign of the negative gradient at $\mathbf{y} = \mathbf{0}$ as an approximation for the minimizing vector of the energy function is that $\mathbf{0}$ is the centroid of all possible bipolar vectors. In this way we approximate the energy function by a hyperplane with the slope of the gradient going through $\mathbf{0}$ and find the minimum bipolar vector on that hyperplane. To see how this plays out with a specific example, consider the RBF kernel $k_y(\mathbf{r}, \mathbf{t}) = \exp(-\gamma \|\mathbf{r} - \mathbf{t}\|^2)$, $\gamma > 0$. Taking the gradient, we obtain

$$\frac{\partial k_{\mathbf{y}}(\mathbf{r}, \mathbf{t})}{\partial \mathbf{r}} = -2\gamma(\mathbf{r} - \mathbf{t}) \exp(-\gamma \|\mathbf{r} - \mathbf{t}\|^2) \quad (13)$$

So then $\mathbf{s}_{\mathbf{y}}(t+1)$ is equal to

$$\text{sgn} \left(\sum_{i=1}^p -2\gamma(\mathbf{y} - \mathbf{y}^{(i)}) \exp(-\gamma \|\mathbf{y} - \mathbf{y}^{(i)}\|^2) k_{\mathbf{x}}(\mathbf{s}_{\mathbf{x}}(t), \mathbf{x}^{(i)}) \right) \quad (14)$$

At $\mathbf{y} = \mathbf{0}$,

$$\mathbf{s}_{\mathbf{y}}(t+1) = \text{sgn} \left(\sum_{i=1}^p \mathbf{y}^{(i)} \exp(-\gamma \|\mathbf{y}^{(i)}\|^2) k_{\mathbf{x}}(\mathbf{s}_{\mathbf{x}}(t), \mathbf{x}^{(i)}) \right) \quad (15)$$

Since the $\mathbf{y}^{(i)}$ are bipolar, and the norm of all bipolar vectors of a given dimension n is \sqrt{n} , it turns out that $\exp(-\gamma \|\mathbf{y}^{(i)}\|^2) = \exp(-\gamma n)$ is a positive constant for all i and we can simplify to

$$\mathbf{s}_{\mathbf{y}}(t+1) = \text{sgn} \left(\sum_{i=1}^p \mathbf{y}^{(i)} k_{\mathbf{x}}(\mathbf{s}_{\mathbf{x}}(t), \mathbf{x}^{(i)}) \right) \quad (16)$$

With the specific choice $k_{\mathbf{x}}(\mathbf{r}, \mathbf{t}) = \alpha^{\mathbf{r}^T \mathbf{t}}$, $\alpha > 1$, this is the update equation in the exponential BAM (eBAM) [13]; in the autoassociative case, it is equivalent to the kernel formalization of the RCAM as presented in [14]. This is interesting because it gives a unifying perspective on the kernelized recall functions of these other works, and a baseline for comparison against the other inference methods proposed here.

The previously described update step can be improved upon in a simple manner by noting that $\mathbf{0}$ is only one of many possible start points for taking the gradient. In what we call the *stochastic* BAM one considers several random starting points with components in $[-1, +1]$, taking the sign of the gradient as before, and then choosing the resulting vector that gives the lowest value for the energy function. To ensure that this gives a vector with a lower energy than would result from the previous method, one must always include $\mathbf{0}$ as one of the possible starting vectors. In practice, this performs better than the naive method, which shows the advantage of kernelizing the energy function rather than the inference method directly.

Other approaches entail treating the problem directly as a discrete optimization problem. The start point chosen is the sign of the gradient of the energy function at $\mathbf{0}$. From here, one iterates over each component of the vector in a random order, flipping its value from $+1$ to -1 or vice versa if this flipping improves the overall energy; one continues to do this until no bit can be flipped. A similar approach is to do *steepest descent* hill-climbing: rather than flipping any bit that improves the energy, one checks every component and changes the component that *most* decreases the energy.

4 EXPERIMENTAL WORK

The experimentation part of this work is divided into two sections, with the first exploring the associative memory properties of the kernel BAM with various inference methods and the second displaying additional practical applications of the kernel BAM.

4.1 A study in capacity and noise

The first study focuses on the associative memory properties of the kernel BAM, namely *capacity* and *noise performance*. Each of these

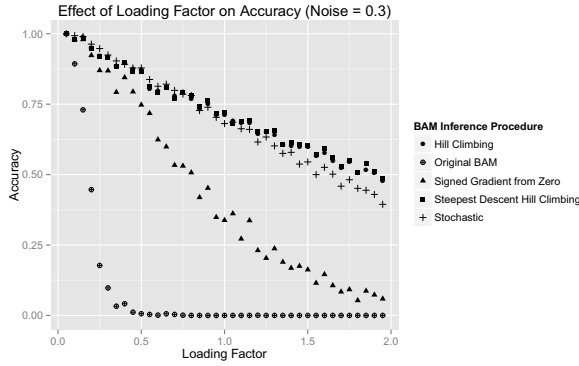
properties is tested with three different kernels: polynomial kernels of degree 2 and 3, and the RBF kernel. For the latter, $\gamma = 0.5$. In each chart the four different inference procedures presented in Section 3.2 are compared. In the case of the stochastic inference procedure, 10 random start points are performed. ‘Loading factor’ signifies the fraction of patterns stored with respect to the dimension of the patterns. Noise is also written as a fraction of the dimension of the data. For the experiments shown, the dimensionality of both $\mathbf{x}^{(i)}$ and $\mathbf{y}^{(i)}$ was $m = n = 32$. For space reasons, only a sample of the results will be shown. The tests were performed as follows:

For each loading factor $\psi \in \{0.05, 0.1, 0.15, \dots, 1.95\}$:

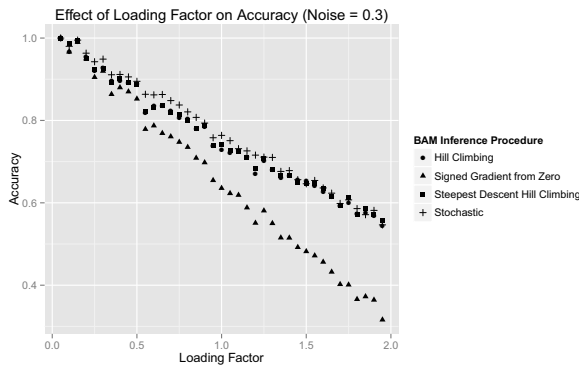
- Generate five datasets $\mathcal{S}_1, \dots, \mathcal{S}_5$ of size $p = n \cdot \psi$ where n is the dimension of the input patterns. Each dataset consists of p randomly generated unique pairs $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ of bipolar vectors.
- For each noise amount $\eta \in \{0, 0.05, 0.1, \dots, 0.5\}$, generate new test datasets \mathcal{S}'_j by, for each vector $\mathbf{x}^{(i)}$ in \mathcal{S}_j , adding 10 corresponding noisy vectors $\hat{\mathbf{x}}_q^{(i)}$, $q \in \{1, \dots, 10\}$ by randomly flipping exactly $\eta \cdot n$ bits of $\mathbf{x}^{(i)}$ 10 times. Thus, for all $j \in \{1, \dots, 5\}$, $\mathcal{S}'_j = \{(\hat{\mathbf{x}}_q^{(i)}, \mathbf{y}^{(i)}) \mid q \in \{1, \dots, 10\}, i \in \{1, \dots, p\}\}$
- For each *bam* in the BAMs to test; for $j \in \{1 \dots 5\}$, train *bam* with \mathcal{S}_j ; for each $(\hat{\mathbf{x}}_q^{(i)}, \mathbf{y}^{(i)}) \in \mathcal{S}'_j$, use *bam* to recall $\hat{\mathbf{y}}^{(i)}$ with the noisy $\hat{\mathbf{x}}_q^{(i)}$ as the input. If $\hat{\mathbf{y}}^{(i)} = \mathbf{y}^{(i)}$ exactly, the recall is considered correct
- Compute the average accuracies as $\text{correct_recalls}/\text{total_tests}$ for each *bam* over all five datasets

Capacity. We first compare how the various recall methods perform with respect to the loading factor and the kernel. The performance of the original, non-kernelized BAM [2] is included in the figures of the degree-2 polynomial kernel for reference. Figure 3 displays the effect of the loading factor on recall performance when noise is fixed at 0.3. It is first noticeable that the biggest difference in performance comes with the polynomial kernels as seen in Figs. 3a and 3b, and also that the general performance of the BAMs increases from the degree-2 to the degree-3 polynomial and from the degree-3 polynomial to the RBF –Fig. 3c. This can be explained by the fact that the corresponding feature spaces are increasing in dimension (with the one in the RBF kernel being infinite-dimensional), doing a better job at orthogonalizing the input patterns in feature space. Note that in most cases, the ‘signed gradient from zero’ method performs worse than the other optimization-based inference methods, especially for the polynomial kernels. This shows the advantage of kernelizing the energy function and operating the BAM in feature space rather than kernelizing the inference procedure directly. It also appears that –among the more advanced inference methods– those based on hill-climbing work best for the degree-2 polynomial, while the stochastic one seems to work better for the other kernels. This could be due to the fact that hill-climbing is likely to get caught in local minima, so when the energy surface is more bumpy, as might be the case with the RBF kernel, the stochastic procedure can avoid the local minima that trap the hill-climbing procedures.

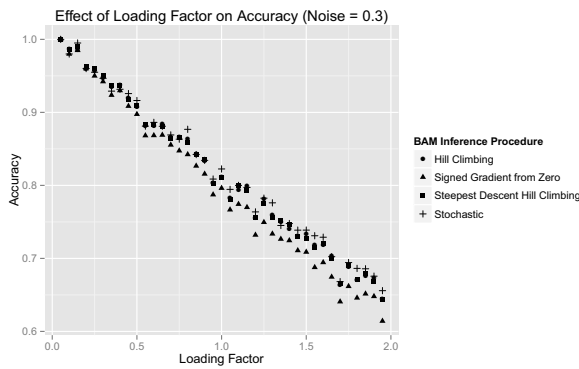
In contrast, with a smoother energy surface, like that of the degree-2 polynomial, hill-climbing is more effective in finding a global minimum than the stochastic procedure. It is also notable that accuracy seems to decrease approximately linearly with an increase in loading factor for all four proposed inference algorithms, with the accuracy of the original BAM decreasing at a rate that is exponential in appearance. The original BAM performs much worse than all other methods in virtually all cases.



(a) Polynomial kernel (degree 2)



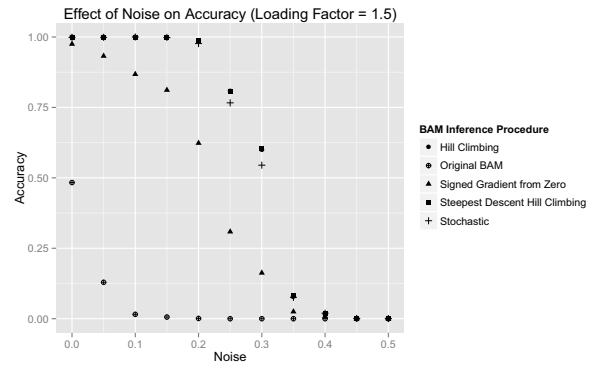
(b) Polynomial kernel (degree 3)



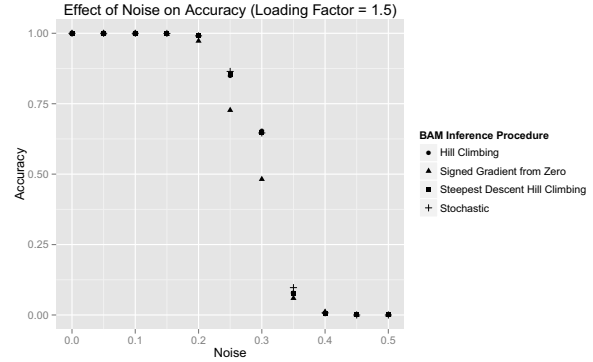
(c) RBF kernel

Figure 3: Effect of loading factor on performance ($\eta = 0.3$).

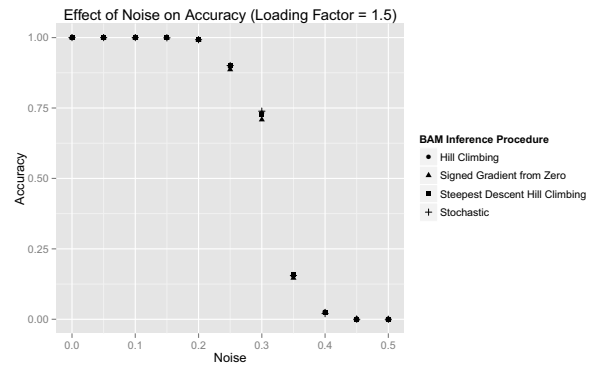
Noise performance. The comments on performance with respect to capacity apply to performance with respect to noise as well, as seen in Fig. 4. With a loading factor of 1.5, the performance difference between inference methods is again most noticeable with the degree-2 polynomial. However, the difference between the inference algorithms is less marked on the endpoints, where accuracy is either very low or very high for all of them. Again, the RBF kernel performs better in general than the other two kernels. Up to 15% noise, all kernel algorithms yield perfect accuracy, even with a loading factor of 1.5. In contrast, even with 0% noise, the original BAM only obtains an accuracy of around 45%. With a loading factor of 0.2 (not shown), all proposed algorithms have a near perfect accuracy with up to 25% noise.



(a) Polynomial kernel (degree 2)



(b) Polynomial kernel (degree 3)

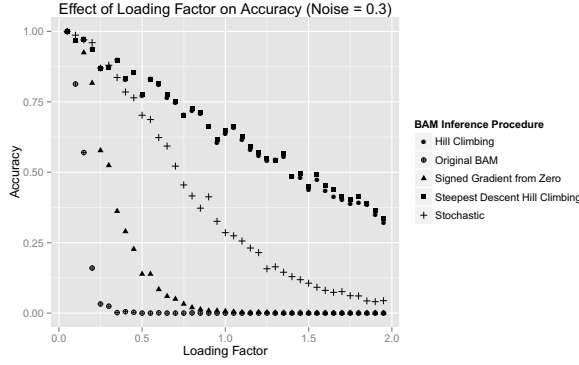


(c) RBF kernel

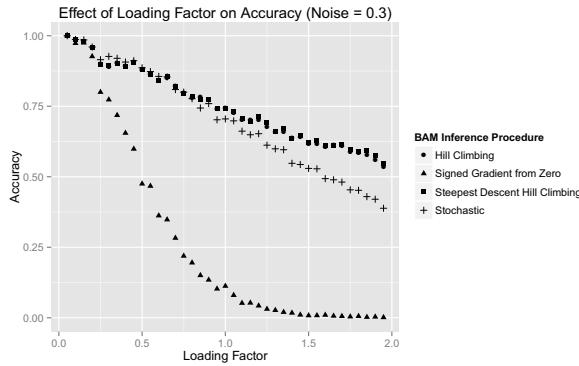
Figure 4: Effect of noise on performance ($\psi = 1.5$).

4.2 A study in one-shot behavior

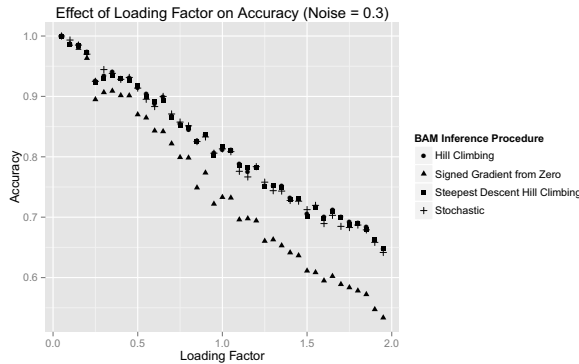
This second study focuses on the “one-shot” performance of the kernel BAM: how it performs when only *one* update step is allowed. The one-shot performance has consequences for applications like classification, where a vector is associated with a class, and only one update step is used to predict it. It also gives an idea of the effectiveness of one update step for a given recall method.



(a) Polynomial kernel (degree 2)



(b) Polynomial kernel (degree 3)

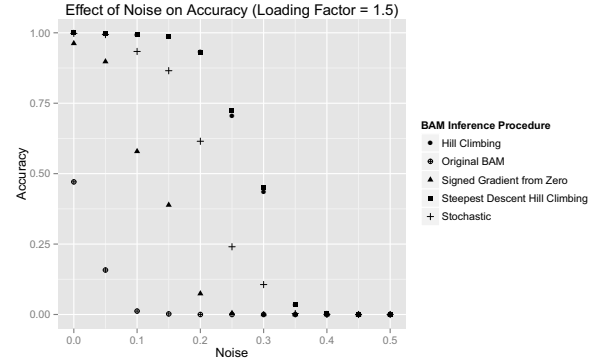


(c) RBF kernel

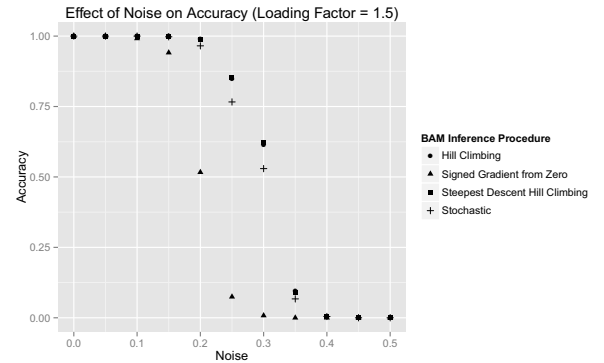
Figure 5: Effect of loading factor on one-shot performance ($\eta = 0.3$).

Figs. 5 and 6 show the accuracy of the various recall methods when only one iteration is allowed. Other than that, the experiments were performed in exactly the same manner as before. In this one-shot scenario, it turns out that differences between the new optimization-based recall methods and the simplest gradient-based method are

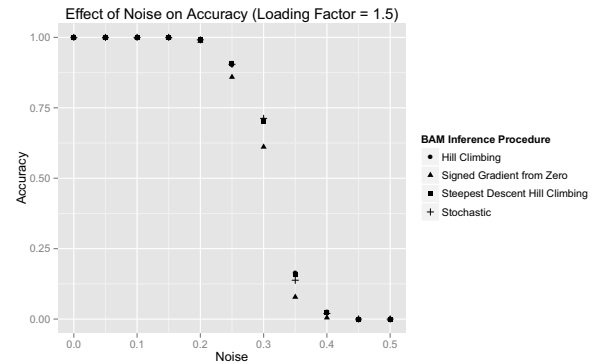
even more pronounced. For example, in Fig. 5a, the accuracy of hill-climbing methods seem to only decline linearly with respect to the increasing loading factor, whereas the simple method seems to decline exponentially, reaching an accuracy of 0% at a loading factor of 1.0 while the hill-climbing methods still have an accuracy of around 63%. Even with the RBF kernel, the difference is significant. The stochastic method is also markedly worse than the hill-climbing methods in the degree-2 polynomial, where in the case with unlimited iterations the performance difference is not particularly noteworthy. Similar comments can be made with respect to the noise performance in Fig. 6.



(a) Polynomial kernel (degree 2)



(b) Polynomial kernel (degree 3)



(c) RBF kernel

Figure 6: Effect of noise on one-shot performance ($\psi = 1.5$).

This demonstrates further that recall techniques that specifically target the minimization of the energy function and thus better main-

tain the analogy with the original BAM are much more effective in each update step, even in the case of the RBF kernel where after many iterations the performance is more similar.

5 APPLICATIONS

This section presents some more practical applications of the kernel BAM. First, the performance of the kernel BAM is tested on associations between images (heteroassociation). For this, we used the MNIST dataset², which consists of 60,000 training images and 10,000 test images of size 28x28 pixels, each valued between 0 and 255. The images were linearly rescaled to the interval $[-1, +1]$. This made selecting the RBF parameter easier, since the parameter $\gamma = 0.5$ seems to work well for bipolar vectors in general³.

Image association. First we created an association list between images without any a priori relation. We randomly selected one example of each digit 0-9, binarized each image, and created a kernel BAM with the RBF kernel for both k_x and k_y (with $\gamma = 0.5$) to store associations between each digit and the digit following it. In other words, the associations stored were $\{('0', '1'), ('1', '2'), \dots, ('9', '0')\}$ —see Fig. 7a. The simplest inference procedure was used to show the recovery of images under two types of noise. In one test, salt and pepper noise was added by randomly flipping 20% of the bits in the input image—Fig. 7b. Since all of these images are correlated (white in the middle, black around the borders), recall is made more difficult than with randomly chosen bipolar vectors. Still, all of these patterns were recovered perfectly. With more than 20% noise, recall performance began to suffer, even with the other more advanced inference methods.

Image reconstruction. In the next test, the top half of the input image was blacked out—Fig. 7c. All associated images were correctly recalled, despite some of the noisy inputs looking very similar. For example, the '4' and the '9' look very similar with the top half blacked out, as do the '1' and the '7'. This could have interesting applications in image completion.

Image generation. It is perfectly possible to create an association between an image and a "class" that is also an image. Since there are far fewer classes (the numbers 0–9) than images, performing one-shot recall using a class as input will output the image giving the lowest energy for that class, even if it is not an actual stored image; in other words, the most 'typical' image as learned from the data in the BAM. To test this, we randomly sampled 200 images from each class, and used them to train the kernel BAM with the simple, stochastic, and hill-climbing recall procedures. Class labels were encoded by using the binary representation of their ASCII character. The results can be seen in Fig. 8. The obvious comment is that the stochastic BAM is wrong 5 out of 10 times in what it draws—Fig. 8b. Why it draws the incorrect digits is unclear, but it certainly reveals the random nature of the algorithm. It could be that the stochastic BAM retrieves an actual stored memory, albeit one of the wrong class, that happens to have a lower energy than the image displayed by the simple BAM, which seems to be a kind of aggregate of other stored memories. The simple and the hill-climbing inference procedure both correctly draw every digit. The digits written with the hill-climbing technique tend to be more clear than those of the simple inference procedure (e.g. the hole is present in the bottom of the '8').

Image classification. Because the kernel BAM is heteroassociative, it may be used for classification problems, storing associations between individuals and their classes. One-shot recall may then be

used to predict the class of an input. In this case it would be possible for the kernel BAM to work on continuous input data spaces.

In this task, we also used the USPS Handwritten Digits dataset, in which the original scanned digits are binary and of different sizes and orientations; the images here have been deslanted and size normalized, resulting in 16x16 grayscale images. We tested classification not only with the simple kernel BAM and the stochastic kernel BAM but also with a few other algorithms for comparison. First, we compare with a nearest-neighbor classifier (NNC) that uses Euclidean distance. This is a standard classifier to compare to, as it is easy to implement and performs well despite its simplicity. Next, since there are just a few classes, it is possible to simply pick the class that gives the lowest energy for the input image. This is what the inference methods are trying to do in the first place and corresponds to the best possible case for recall using a kernel BAM; we call this algorithm the Lowest-energy BAM or LeBAM. Lastly, we compare to the method presented by Caputo et al. [8] because their kernel energy function is the same in the autoassociative case as that of the kernel BAM. They propose to build a kernel *autoassociative* memory for each class, each using the images for that class only. The class of an input is predicted by computing the energy of each AM for that input and then choosing the class that gives the lowest energy.

The results of classification on both the binarized and continuous versions of the input may be seen in Table 1. For all of the BAMs as well as for Caputo's method, an RBF kernel was used with $\gamma = 0.5$, for both k_x and k_y . In almost all cases, the kernel AMs performed comparatively well against NNC. On both datasets, the LeBAM, which can be considered as the ideal kernel BAM, gives the globally best results, though it performs the same on the USPS dataset as the other two BAM methods. The fact that all kernel AMs obtain competitive accuracies in a supervised classification task for which they were not conceived is also remarkable.

Table 1: Test accuracies for the MNIST and USPS data. The value on the left corresponds to the accuracy when using binarized images, whereas the one on the right corresponds to the accuracy when using continuous values for the pixels (all values are percentages).

	NNC	BAM	Stoch. BAM	LeBAM	ref. [8]
MNIST	96.14 - 96.91	96.13 - 96.95	96.16 - 96.95	96.16 - 96.96	96.16 - 96.95
USPS	92.92 - 94.37	93.07 - 94.57	93.07 - 94.57	93.07 - 94.57	93.07 - 94.52

6 CONCLUSIONS AND FUTURE WORK

In this work, a new bidirectional associative memory (BAM) has been presented that uses the kernel trick to improve capacity and noise performance. Unlike other kernel associative memories, many of which kernelize the inference method of the associative memory directly, the kernel BAM presented here kernelizes the energy function, which allows for the creation of more effective recall procedures. Experimental work on synthetic data strongly suggests the effectiveness of the new recall procedures, and several practical use cases have been demonstrated using synthetic and real data.

The kernel BAM yields many different directions for future work, some of which have been mentioned previously. One opportunity for improvement would be to adapt the kernel BAM to work on continuous data, which could be achieved by replacing the discrete optimization methods presented by continuous ones, though it would be still necessary to constrain the domain of the input vectors (to vectors of length 1, for example, or with components in the interval $[-1, +1]$).

² <http://yann.lecun.com/exdb/mnist/>

³ See the Appendix for an explanation of this fact.



(a) Original image pairs. Each column represents a pair to be stored; i.e., each image in the top row is associated with the image directly below it.



(b) Recovery with 20% random salt and pepper noise. The first row shows the inputs to the network and the second shows the output for each input.



(c) Recovery with the top 50% of the image blacked out. The first row shows the inputs to the network and the second shows the output for each input.

Figure 7: Associating a number with the number following it.



(a) Simple Inference Procedure



(b) Stochastic Inference Procedure



(c) Hill-climbing Inference Procedure

Figure 8: Image generation with the kernel BAM. The RBF kernel was used for both k_x and k_y , with $\gamma = 0.5$.

One interesting thing to note about the hill-climbing approaches is that, since they do not necessarily require the gradient of the energy function (barring the choice of the start point, which could be done differently), it would be possible to use non-differentiable kernel functions for k_x and k_y . This would allow for the use of the kernel BAM to store non-numeric data types, like text or graphs, for which working kernels exist [15, 16, 17]. The difficulty in this would be to find better optimization methods for the update step that would accommodate different kinds of data. However, classification using the described LeBAM approach would already be straightforward with kernels on non-numeric data, since only the energy function is required and not an actual update step, demonstrating another benefit of kernelizing the energy function.

A Choosing the RBF parameter for the Kernel BAM

Noting that we can write $\gamma = 0.5 \cdot \sigma^{-2}$, where σ^2 is the variance of a Gaussian, we see that $\gamma = 0.5$ corresponds to $\sigma = 1$. Looking at the plot of the energy function in Fig. 1a (which happens to be the real energy function of a kernel BAM with an RBF kernel), we can see that each basin of attraction can be put in correspondence to a Gaussian curve, and the basins of attraction of two adjacent memories meet around two standard deviations away from the center of their respective Gaussians. When we consider that—in a bipolar vector—the smallest possible change is to flip one component from ± 1 to ∓ 1 , taking σ to be somewhere between 0.5 and 2 makes sense and we chose to fix $\sigma = 1$ as the geometric mean of these values.

REFERENCES

- [1] J.J. Hopfield, 'Neural networks and physical systems with emergent collective computational abilities', *Proceedings of the national academy of sciences*, **79**(8), 2554–2558, (1982).
- [2] B. Kosko, 'Bidirectional associative memories', *Systems, Man and Cybernetics, IEEE Transactions on*, **18**(1), 49–60, (1988).
- [3] M.E. Acevedo-Mosqueda, C. Yáñez-Márquez, and M.A. Acevedo-Mosqueda, 'Bidirectional associative memories: Different approaches', *ACM Computing Surveys*, **45**(2), 18:1–18:30, (2013).
- [4] D. Nowicki and H. Siegelmann, 'Flexible kernel memory', *PloS One*, **5**(6), (2010).
- [5] S. Chartier, M. Boukadoum, and M. Amiri, 'BAM learning of nonlinearly separable tasks by using an asymmetrical output function and reinforcement learning', *Neural Networks, IEEE Transactions on*, **20**(8), 1281–1292, (2009).
- [6] S. Chartier, G. Giguere, P. Renaud, J.M. Lina, and R. Proulx, 'FEBAM: A feature-extracting bidirectional associative memory', in *Neural Networks, 2007. IJCNN 2007. International Joint Conference on*, pp. 1679–1684, (2007).
- [7] R. Rojas, *Neural Networks: A Systematic Introduction*, Springer Berlin Heidelberg, 1996.
- [8] B. Caputo and H. Niemann, 'From markov random fields to associative memories and back: Spin glass markov random fields', *Proc. of IEEE Workshop on Statistical and Computational Theories of Vision*, 101–102, (2001).
- [9] C. García and J.A. Moreno, 'The Hopfield associative memory network: Improving performance with the kernel "trick"', in *Advances in Artificial Intelligence–IBERAMIA 2004*, 871–880, Springer, (2004).
- [10] S. Chen, L. Chen, and Z.H. Zhou, 'A unified SWSI–KAMS framework and performance evaluation on face recognition', *Neurocomputing*, **68**, 54–69, (2005).
- [11] B. L. Zhang, H. Zhang, and S.S. Ge, 'Face recognition by applying wavelet subband representation and kernel associative memory', *Neural Networks, IEEE Transactions on*, **15**(1), 166–177, (2004).
- [12] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang, 'A tutorial on energy-based learning', Bakir et al. (eds), *Predicting Structured Outputs*, MIT Press (2006).
- [13] Y.J. Jeng, C.C. Yeh, and T.D. Chiueh, 'Exponential bidirectional associative memories', *Electronics Letters*, **26**(11), 717–718, (1990).
- [14] R. Perfetti and E. Ricci, 'Recurrent correlation associative memories: a feature space perspective', *Neural Networks, IEEE Transactions on*, **19**(2), 333–345, (2008).
- [15] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins, 'Text classification using string kernels', *The Journal of Machine Learning Research*, **2**, 419–444, (2002).
- [16] T. Gärtner, P. Flach, and S. Wrobel, 'On graph kernels: Hardness results and efficient alternatives', in *Learning Theory and Kernel Machines*, 129–143, Springer, (2003).
- [17] T. Gärtner, 'A survey of kernels for structured data', *ACM SIGKDD Explorations Newsletter*, **5**(1), 49–58, (2003).